MONGODB MODELING

Phó Nghĩa Văn

GIỚI THIỆU PROECT

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

timviecit.com

Modeling trong dự án thực tế

- Các bước xây dựng một project: Từ đặc tả (Software Requirement Specification -SRS) --> phân tích UML --> thiết kế database --> Code
- Khi nào sử dụng MongoDB?
  - Khi hệ thống realtime, yêu cầu phản hồi nhanh
  - Hệ thống Big data yêu cầu truy vấn nhanh
  - Khi đặc tả chưa rõ ràng hoặc không thể thiết database hoàn chỉnh ngay từ ban đầu
  - Khi yêu cầu xây dựng ứng dụng nhanh, loại dữ liệu đa dạng
  - Khi hệ thống cần phải chịu nhiều truy cập cùng một lúc (tính chịu tải)

CÁC MỐI QUAN HỆ

- Quan hệ 1 - 1 (one to one)
  - Referencing documents
  - Embedding documents
- Quan hệ 1 - N (one to many)
  - Referencing documents
  - Embedding documents
  - Bucketing
- Quan hệ N - M (many to many)
  - One way referencing
  - Two way referencing

EMBEDDING

- Tên khác: denormalization
- Giả sử 1 post có 100 comments
- Cấu trúc: tất cả dữ liệu được lưu thành 1 file trên ổ cứng
- Ưu điểm:
  - Chỉ cần query 1 lần (tương ứng với 1 lần đọc file trên ổ cứng) có thể lấy được 1 post + 100 comments
  - Chỉ sử dụng khi số lượng sub-document trên dưới 100

```
{
  title: "An awesome blog",
  url: "http://awesomeblog.com",
  text: "This is an awesome blog we have just started",
  comments: [{
    name: "Peter Critic",
    created_on: ISODate("2014-01-01T10:01:22Z"),
    comment: "Awesome blog post"
  }, {
    name: "John Page",
    created_on: ISODate("2014-01-01T11:01:22Z"),
    comment: "Not so awesome blog"
  }]
}
```

EMBEDDING

- Nhược điểm:
  - Khi số lượng comments quá lớn sẽ vượt quá **maximum size** mặc định của 1 document (16Mb) (mongodb lưu dữ liệu trên ổ cứng ở dạng BSON - binary JSON, và có kích thước tối đa mặc định là 16Mb)
  - **Không thể thực hiện pagination với performance tốt**, vì mỗi lần query bằng phương thức find() thông thường sẽ lấy luôn tất cả các comments có trong một bài post

```
{
  title: "An awesome blog",
  url: "http://awesomeblog.com",
  text: "This is an awesome blog we have just started",
  comments: [{
    name: "Peter Critic",
    created_on: ISODate("2014-01-01T10:01:22Z"),
    comment: "Awesome blog post"
  }, {
    name: "John Page",
    created_on: ISODate("2014-01-01T11:01:22Z"),
    comment: "Not so awesome blog"
  }]
}
```

REFERENCING

- Tên khác: linking, normalization
- Giả sử 1 post có 1000 comments
- Cấu trúc: dữ liệu được lưu thành 1 + 1000 file trên ổ cứng
- Ưu điểm:
  - Khi số lượng comments tăng đột biến (lên đến 1000) thì vẫn không vượt quá maximum size dành cho mỗi file trên ổ cứng
  - Dùng khi số lượng "sub-document" quá lớn

```
{
  _id: 1,
  title: "An awesome blog",
  url: "http://awesomeblog.com",
  text: "This is an awesome blog we have just started"
}
```

```
{
  blog_entry_id: 1,
  name: "Peter Critic",
  created_on: ISODate("2014-01-01T10:01:22Z"),
  comment: "Awesome blog post"
}
{
  blog_entry_id: 1,
  name: "John Page",
  created_on: ISODate("2014-01-01T11:01:22Z"),
  comment: "Not so awesome blog"
}
```

REFERENCING

- Nhược điểm:
  - Mỗi lần query 1 post + 1000 comments (tương ứng với việc đọc 1 file post + 1000 file comments trên ổ cứng) --> máy tính phải hoạt động nhiều

```
{
  _id: 1,
  title: "An awesome blog",
  url: "http://awesomeblog.com",
  text: "This is an awesome blog we have just started"
}
```

```
{
  blog_entry_id: 1,
  name: "Peter Critic",
  created_on: ISODate("2014-01-01T10:01:22Z"),
  comment: "Awesome blog post"
}
{
  blog_entry_id: 1,
  name: "John Page",
  created_on: ISODate("2014-01-01T11:01:22Z"),
  comment: "Not so awesome blog"
}
```

BUCKETING (HYBRID)

- Kỹ thuật bucketing (hybrid) là sự kết hợp ưu điểm của embedding và Referencing
- Các comments được embed vào post theo các buckets, mỗi bucket chứa khoảng 50 comments (số lượng có thể tùy ý thay đổi)

```
{
  blog_entry_id: 1,
  page: 1,
  count: 50,
  comments: [{
    name: "Peter Critic",
    created_on: ISODate("2014-01-01T10:01:22Z"),
    comment: "Awesome blog post"
  }, ...]
}
{
  blog_entry_id: 1,
  page: 2,
  count: 1,
  comments: [{
    name: "John Page",
    created_on: ISODate("2014-01-01T11:01:22Z"),
    comment: "Not so awesome blog"
  }]
}
```

PLAYGROUND

MÔ PHỎNG
CÁC MỐI QUAN HỆ

QUAN HỆ 1 - 1

- Một user có một profile, một profile chỉ thuộc một user
- Có 2 cách để mô phỏng mối quan hệ này:
  - Embedding
  - Referencing

QUAN HỆ 1 - 1

REFERENCING

- Tạo Schema User và Model
- **Profile** thực hiện **ref** đến User thông qua **userId**
- **Chú ý**: do demo cách kỹ thuật referencing nên chỉ tạo một số thuộc tính cơ bản

```
// create User Schema and Model
const UserSchema = new mongoose.Schema({
    username: {type: String, required: true},
    password: {type: String, required: true}
})
const User = mongoose.model('User', UserSchema);


// create Profile Schema and Model
const ProfileSchema = new mongoose.Schema({
    userId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
    university: {type: String, required: true},
    major: {type: String, required: true}
})
const Profile = mongoose.model('Profile', ProfileSchema);
```

QUAN HỆ 1 - 1

REFERENCING

- Tạo instance newUser và newProfile

```
// Create instance
const newUser = new User({
    username: "Hackagon",
    password: "hackagon"
})
newUser.save()
    .then(console.log)
    .catch(console.log)


const newProfile = new Profile({
    userId: newUser._id,
    university: "Cybersoft Academy",
    major: "Fullstack Web Development"
})
newProfile.save()
    .then(console.log)
    .catch(console.log)
```

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

QUAN HỆ 1 - 1

REFERENCING

```
Profile
    .find()
    .populate('userId', 'username password -_id')
    .then(console.log)
    .catch(console.log)
```

- Sử dụng phương thức **populate** để lấy các thông tin cần thiết của user thông qua userId (ObjectId) từ profile

```
[ {  _id: 5c08fa92a5422858f5e86e4b,
    userId: { username: 'Hackagon', password: 'hackagon' },
    university: 'Cybersoft Academy',
    major: 'Fullstack Web Development',
    __v: 0 } ]
```

QUAN HỆ 1 - 1

EMBEDDING

- Tạo ProfileSchema
- Tạo UserSchema
- Trong UserSchema nhúng (embed) vào ProfileSchema

```
// create Profile Schema and Model
const ProfileSchema = new mongoose.Schema({
    university: {type: String, required: true},
    major: {type: String, required: true}
})
const Profile = mongoose.model('Profile', ProfileSchema);

// create User Schema and Model
const UserSchema = new mongoose.Schema({
    username: {type: String, required: true},
    password: {type: String, required: true},
    profile: {
        type: ProfileSchema,
        required: true
    }
})
const User = mongoose.model('User', UserSchema);
```

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

QUAN HỆ 1 - 1

EMBEDDING

- Tạo instance newProfile
- Tạo instance newUser
- instance newUser sử sụng instance newProfile như một giá trị của thuộc tính profile

```
// Create instance
const newProfile = new Profile({
    university: "Cybersoft Academy",
    major: "Fullstack Web Development"
})

const newUser = new User({
    username: "Hackagon",
    password: "hackagon",
    profile: newProfile
})
newUser.save()
    .then(console.log)
    .catch(console.log)
```

QUAN HỆ 1 - 1

EMBEDDING

- Thực hiện query User
- Ta sẽ được thông tin về profile của User
- **Ưu điểm:** chỉ cần query 1 lần, có được cả thông tin của user và profile tương ứng

```
User
    .find()
    .then(console.log)
    .catch(console.log)
```

```
[ { _id: 5c09e6f0db40fc1cee4d55d9,
  username: 'Hackagon',
  password: 'hackagon',
  profile:
   { _id: 5c09e6f0db40fc1cee4d55d8,
     university: 'Cybersoft Academy',
     major: 'Fullstack Web Development' },
  __v: 0 } ]
```

QUAN HỆ 1 - N

- Một bài post có nhiều comment, một comment chỉ thuộc một bài post
- Có 3 cách để mô phỏng mối quan hệ này:
  - Embedding (tương tự quan hệ 1 - 1)
  - Referencing (tương tự quan hệ 1 - 1)
  - Bucketing

QUAN HỆ 1 - N

- Tạo Schema cho Post và Comment tương tự như referencing

```
const PostSchema = new mongoose.Schema({
    title: { type: String, required: true },
    content: { type: String, required: true }
})

const Post = mongoose.model('Post', PostSchema);

const CommentSchema = new mongoose.Schema({
    postId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Post'
    },
    content: { type: String, required: true }
})

const Comment = new mongoose.model('Comment', CommentSchema);
```

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

QUAN HỆ 1 - N

- Tạo các instance cho post và comments
- Save các instance

```
const post = new Post({
    title: 'Hello world',
    content: "this is the first lesson"
})

const comment1 = new Comment({
    postId: post._id,
    content: 'content 01'
})
const comment2 = new Comment({
    postId: post._id,
    content: 'content 02'
})
const comment3 = new Comment({
    postId: post._id,
    content: 'content 03'
})
const comment4 = new Comment({
    postId: post._id,
    content: 'content 04'
})
const comment5 = new Comment({
    postId: post._id,
    content: 'content 05'
})
```

QUAN HỆ 1 - N

- Thực hiện query

```
Comment.aggregate()
    .facet({
        post: [
            { $match : { postId : mongoose.Types.ObjectId("5c09f2edb11a8824f9dfe916") } },
            { $bucketAuto:
                {
                    groupBy: '$postId',
                    buckets: 2,
                    output: {
                        comments: {$push: { content:'$content' }}
                    }
                }
            },

        ]
    })
    .exec()
    .then(comments => console.log(JSON.stringify(comments, undefined, 2)))
```

id của bài post

Chia đều document cho mỗi bucket

Số lượng document trong mỗi bucket

Lựa chọn nội dung hiển thị

QUAN HỆ N - M

- Một user có thể thuộc nhiều groups, một groups có thể chứa nhiều users
- Có 2 cách để mô phỏng mối quan hệ này:
  - One way referencing
  - Two way referencing
- Cách implement tương tự referencing
- Cách query (two way referencing):
  - Có group._id tìm group, trong group có danh sách id của user --> tìm được các users
  - Có user._id tìm user, trong user có danh sách id của group --> tìm được các groups

PROJECT

SOCIAL NETWORK

ĐẶC TẢ

Xây dựng một ứng dụng mạng xã hội (social network) với các tính năng sau:

- Cho phép **user** đăng ký, đăng nhập vào hệ thống. Yêu cầu user phải cung cấp các thông tin: username, password, email, ...
- Mỗi user có quyền được tạo một và chỉ một **profile** cá nhân để mô tả về bản thân mình: education, job, description, social network
- Mỗi user có quyền post các **bài biết**. Mỗi bài viết có thể được các user khác đọc, **comment**, **like** và dislike
- Mỗi user có thể tự tạo **group** hoặc tham gia một group nào đó

CÁC BƯỚC THỰC HIỆN

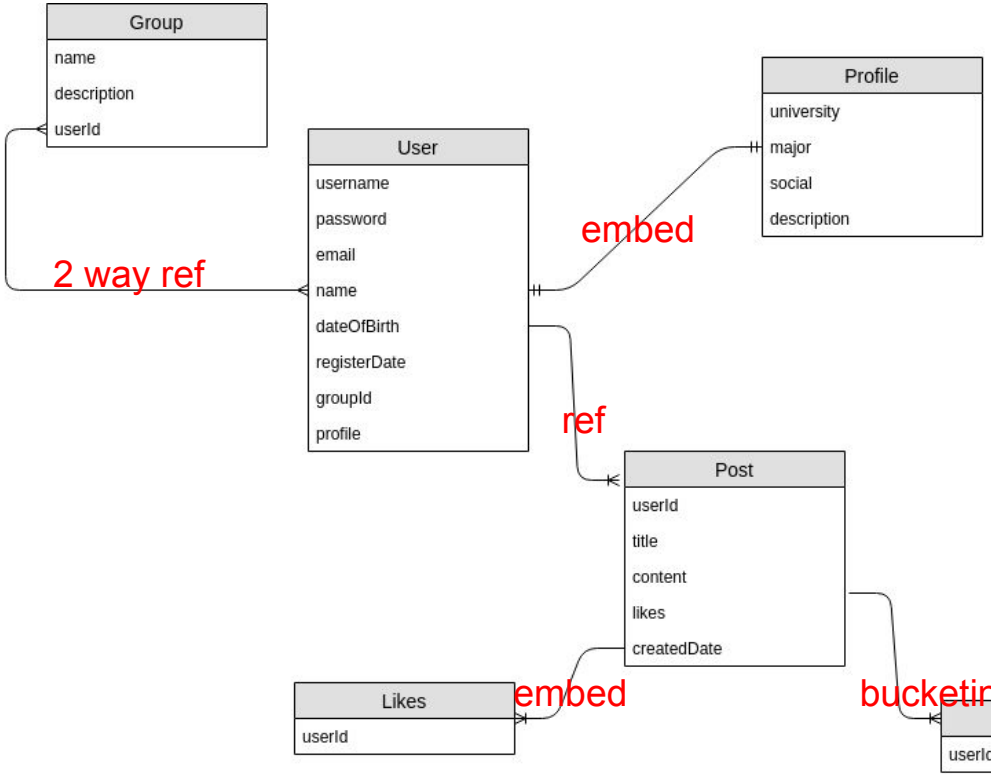
1. Liệt kê các collection và thuộc tính kèm theo
2. Xác định mối quan hệ giữa các collection (1-1, 1-N, N-M) và phương pháp mô hình hóa mối quan hệ đó
3. Code: xây dựng các Schema và model
4. Code: thực hiện các chức năng CRUD cho các model vừa xây dựng được

PHÂN TÍCH ĐẶC TẢ

Từ đặc tả, ta có thể mô hình database với các đối tượng sau:

- **Một** user có **một** profile
- **Một** user có thể có **nhiều** bài post
- **Một** bài post có thể có **nhiều** like
- **Một** bài post có thể có **nhiều** comment
- **Một** user có thể thuộc **nhiều** group
- **Một** group có thể chứa **nhiều** user

SOCIAL NETWORK MODELING



LIVE CODE

LIVE CODE TRỰC TIẾP TẠI LỚP

BÀI TẬP VỀ NHÀ

Thiết kế database cho project xedike: https://xedike.vn/home : nghiên cứu project từ trang web xedike và tự suy nghĩ:

- Viết đặc tả của project
- Thiết kế database: có những collection nào, mối quan hệ giữa các collection là như thế nào
- Xây dựng database bằng nodejs (package mongoose) tương tự như project social network tại lớp